13. Upload Vulnerabilities

📁 Unrestricted File Upload

1. What is an Unrestricted File Upload?

- An **unrestricted file upload** vulnerability occurs when a web application allows users to upload files **without properly validating the file type, content, size, or other security controls**.
- This can lead to attackers uploading **malicious files** like web shells, scripts, or executables, which can compromise the server.

2. Why Are File Uploads a Security Risk?

- File uploads create a direct vector for **remote code execution (RCE)** if malicious scripts get executed on the server.
- Uploaded files can contain **malware, backdoors, or web shells** to maintain persistent access.
- Can lead to **data leakage, privilege escalation**, or complete server takeover.
- Improper handling may allow **denial of service** by uploading large files or flooding storage.

3. How Can File Upload Forms Be Exploited?

- Uploading executable scripts (e.g., PHP, ASP, JSP) disguised as images or documents.
- Bypassing file type checks via **filename obfuscation or double extensions** (e.g., `shell.php.jpg`).
- Exploiting weak MIME-type validation or trusting only client-side checks.
- Uploading files containing **malicious payloads** to be executed or accessed remotely.
- Overwriting critical files or placing files in sensitive directories.

4. What is a Web Shell?

- A **web shell** is a malicious script uploaded to a web server allowing an attacker to execute arbitrary commands remotely via a web interface.
- Common languages: PHP, ASP, JSP, Perl, Python.
- Acts as a backdoor, giving attackers control over the compromised server.

5. How Do MIME Types Relate to Upload Security?

- MIME (Multipurpose Internet Mail Extensions) type specifies the **nature and format of a file** (e.g., `image/png`, `application/pdf`).
- Servers use MIME types to validate uploads by checking if the file content matches the expected type.

- Attackers can spoof MIME types to trick server-side validation if it relies solely on MIME headers.

---

6. What is Content-Type Spoofing?

- Content-Type spoofing is when an attacker **forges or manipulates the MIME type header** sent by the browser or client to bypass server-side validation.
- For example, uploading a PHP script but sending the MIME type as `image/jpeg`.
- Relying only on client-submitted MIME types is insecure.

---

7. How Can Server-Side Validation Mitigate Risks?

- Server-side validation is **mandatory** because client-side checks can be bypassed.
- Techniques include:
  - Checking **file extensions** against a whitelist.
  - Verifying file **content headers (magic bytes)** to confirm file type.
  - Validating file size limits.
  - Renaming files on upload to prevent executable names.
  - Storing uploads outside the web root.

---

8. What is the Importance of File Extension Filtering?

- Filtering file extensions prevents uploading dangerous file types (e.g., `.php`, `.exe`).
- Only allow **safe extensions** like `.jpg`, `.png`, `.pdf`.
- Helps reduce the risk of malicious script execution if combined with other checks.
- Note: Attackers may try double extensions or Unicode tricks, so extension checks must be strict.

---

9. How Can Client-Side Checks Be Bypassed?

- Client-side checks (JavaScript or HTML form restrictions) are easily bypassed by:
  - Disabling JavaScript.
  - Using tools like curl, Postman, or Burp Suite to craft custom requests.
  - Manipulating form data or HTTP headers directly.
- Therefore, client-side validation is only for **user experience, not security**.

---

10. What Are Some Secure File Upload Practices?

- Implement **strict server-side validation** (type, size, content).
- Store files **outside the web root** to prevent direct execution.
- Rename files on upload to **avoid executable extensions**.
- Set **secure file permissions** (read-only where possible).
- Use **anti-virus scanning** on uploaded files.

- Limit **file size** to prevent DoS attacks.
- Use **Content Security Policy (CSP)** and web server configs to restrict file execution.

---

11. How Does File Size Limitation Help Security?

- Restricting file size prevents attackers from:
  - Uploading large files that consume disk space (DoS).
  - Using oversized payloads for buffer overflow or application crashes.
- Helps maintain server stability and resource availability.

---

12. What Are the Risks of Storing Files on the Same Domain?

- Files stored in web-accessible directories can be **executed or downloaded directly**, increasing risk.
- If an attacker uploads a web shell, they can access it via the web domain and control the server.
- Files stored on the same domain can expose sensitive data if directory listing is enabled or access controls are weak.

---

13. How Do File Permissions Affect Upload Security?

- File permissions control who can **read, write, or execute** uploaded files.
- Restrictive permissions prevent:
  - Execution of uploaded scripts.
  - Unauthorized file modifications or deletions.
- Correct permissions are critical to minimize exploitation of uploaded content.

---

14. Why Should Upload Directories Not Be Executable?

- Upload directories should have **no execute permissions** to prevent execution of uploaded scripts.
- Disables direct running of files like PHP or shell scripts even if uploaded.
- Adds a strong security layer to stop attackers from exploiting uploaded malicious files.
